Technical Proposal - Keyword Prioritization

Problem statement

The current search functionality faces a critical challenge in effectively prioritizing biologically important keywords that users include in their queries but are not tagged by PollyBert. During searches, not prioritizing these additional biological terms can result in less relevant results. For instance, in a query like "differentially expressed miRNAs in lung cancer," while the NER model identifies "lung cancer," it might miss highlighting "miRNA," which is crucial for better search outcomes. We need our search to not only identify NER-tagged terms but also prioritize other biologically significant terms for more accurate and comprehensive results.

Challenges:

- **Insufficient Biological Context:**
  - Incomplete recognition of relevant biological terms in user queries.
  - PollyBert may overlook key biologically significant terms.
  - Lack of comprehensive understanding.
- **Sub optimal Ranking of Relevant Datasets:**
  - Current search prioritization focuses primarily on NER-tagged terms.
  - Failure to appropriately rank or boost datasets containing untagged but crucial biological terms.

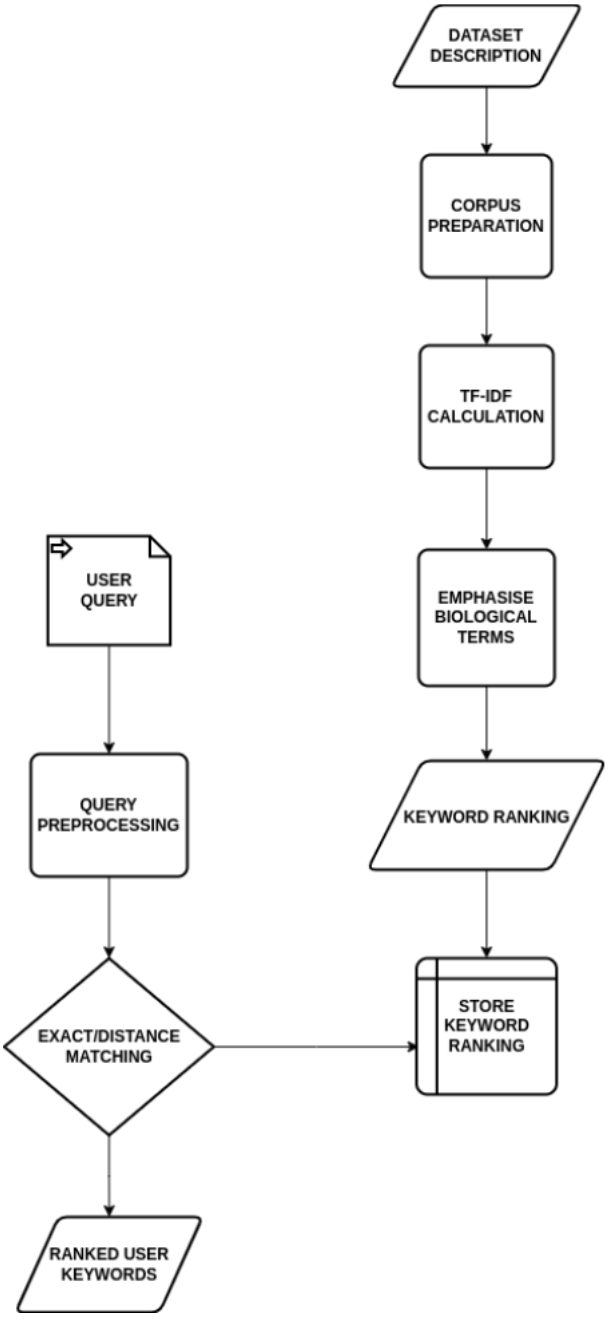
Expected outcomes:

1. **Prioritizing Non-Curated Biological Terms:** By assigning priorities to keywords within the user query, we can effectively identify and rank biologically significant terms, including those that have not been curated.
2. **Keyword Ranking:** The system will employ a TF-IDF keyword ranking algorithm to assign priority flags to the keywords present in user query. This ranking will enable the identification of the most biologically relevant terms within the query.

Objectives

- To improve the parsing of user queries by removing stop words.
- To implement a keyword ranking system for prioritizing biologically relevant keywords.

Implementation details



Corpus Preprocessing:

Fields like **“Description“, “Abstract“, “Overall_design“ and “Summary“** are collected from the dataset level metadata of GEO

```
# Initialize spaCy for text preprocessing
nlp = spacy.load("en_core_web_sm")

#listing out the irrelevant part of speech
irr_tags = ['ADP', 'AUX', 'PUNCT', 'SYM']

# Define a custom tokenizer function
def custom_tokenizer(text):
    doc = nlp(text)
    tokens = [token.lemma_.lower() for token in doc if token.pos_ not in irr_tags and not token.is_stop and not
    return tokens
```

- In the corpus preprocessing step, a custom tokenizer is used to lemmatize and tokenize text using spaCy. Irrelevant parts of speech, stop words, and spaces are removed to extract meaningful keywords

TF-IDF Calculation:

Overview:

- **TF (Term Frequency):** This measures how often a term appears in a particular dataset description. It helps identify the importance of a term within a specific dataset description.
- **IDF (Inverse Document Frequency):** IDF measures the importance of a term across all descriptions in the corpus. It highlights terms that are relatively rare but have significance.

```
tfidf_vectorizer = TfidfVectorizer(
    analyzer='word',
    tokenizer=custom_tokenizer,
    ngram_range=(1, 2),
    sublinear_tf=True,
    smooth_idf=True,
)
```

TF-IDF Parameters:

- `analyzer='word'` : Analyzes words in the text.
- `tokenizer=custom_tokenizer` : Applies the custom tokenizer.
- `ngram_range=(1, 2)` : Considers unigrams and bigrams for context.
- `sublinear_tf=True` : This transformation reduces the impact of very frequent terms. It considers the log-scaled TF, giving less weight to overly frequent terms.
- `smooth_idf=True` : This ensures that terms with zero IDF don't get completely ignored, providing stability to the calculation and preserving the significance of even rare terms.

Keywords are ranked based on their TF-IDF scores.

Example:

Here is a list of top 20 significant scores based on their TF-IDF score (for trial only 10000 significant unique terms are taken from the entire corpus):

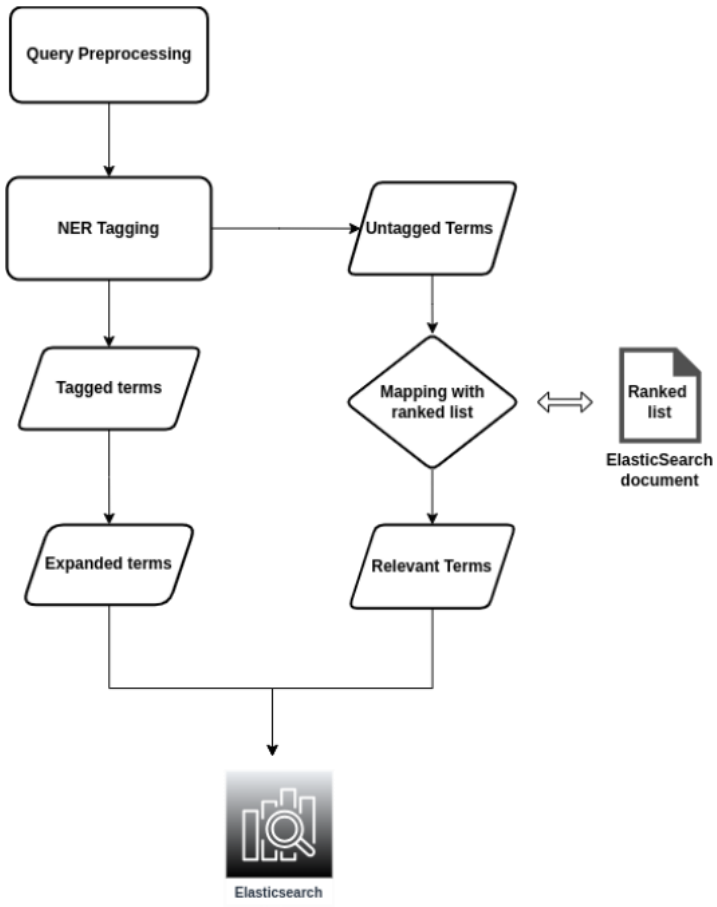
```
Terms : Scores
rna: 0.024604945732437763
cells: 0.024054835161429367
human: 0.02361557519540889
seq: 0.01994373973475735
cell: 0.01922823026052216
rna seq: 0.01759222618474604
genome: 0.016814658246557892
dna: 0.016569766415890985
expression: 0.01583008985603915
integrated: 0.015055606997800518
elements: 0.015014153275372784
human genome: 0.01470117208886496
elements human: 0.01465871196721138
dna elements: 0.014656312965740468
encyclopedia: 0.014650173917817306
encyclopedia dna: 0.014650173917817306
integrated encyclopedia: 0.014650173917817306
analysis: 0.014177970029287904
gene: 0.013422065539293662
cancer: 0.012614477355017716
```

```
Terms : Scores
rna: 0.024604945732437763
cells: 0.024054835161429367
human: 0.02361557519540889
seq: 0.01994373973475735
cell: 0.01922823026052216
rna seq: 0.01759222618474604
genome: 0.016814658246557892
dna: 0.016569766415890985
expression: 0.01583008985603915
integrated: 0.015055606997800518
elements: 0.015014153275372784
human genome: 0.01470117208886496
dna elements: 0.014656312965740468
gene: 0.013422065539293662
cancer: 0.01261447735501771
```

Utilizing GPT model to refine the TF-IDF scores, emphasizing biologically relevant terms

- The score here is a TF-IDF (Term Frequency-Inverse Document Frequency) score for a specific term across the entire corpus.
- The TF-IDF score is calculated by multiplying the TF and IDF scores. It helps identify terms that are both frequent within a specific document and unique across the entire corpus.
- High TF-IDF scores suggest that a term is important.

Code level implementation



Query preprocessing:

- The existing codebase includes a manual list of stop words that are removed from user queries. To enhance this process, we will replace manual stop word removal with spaCy's tokenizer.
- The spaCy tokenizer will efficiently identify and eliminate stop words and other noise from user queries, streamlining the input for further processing

```
nlp = spacy.load("en_core_web_sm")
    # Process the input text
    doc = nlp(text)

    # Define a list of irrelevant part-of-speech tags and stop words
    irrelevant_tags = ["ADP", "AUX", "SYM", "PUNCT"]

    # Iterate through the tokens in the processed text
    for token in doc:
        if token.pos_ in irrelevant_tags or token.is_stop:
            irrelevant_terms.append(token.text)
        else:
            relevant_terms.append(token.text)

#example
text = "treated vs untreated condition in her2+ve breast cancer"
#output
relevant_terms ['treated', 'untreated', 'condition', 'her2+ve', 'breast', 'cancer']
irrelevant_terms: ['vs', 'in']
```

Time taken: 10.7 µs

Identification of untagged terms

```
#Example code
def identify_untagged_terms(user_query, ner_entities):
    #assuming ner_entities are the NER tagged terms

    untagged_terms = [term for term in processed_query if term not in ner_entities]
    return untagged_terms

```

Mapping with ranked list

```
#Example code
def search_additional_terms(index, untagged_terms):
    # Query ElasticSearch for additional relevant terms

    query_body = {"query": {"untagged_terms": {"term": untagged_terms}}}
    results = es.search(index=index, body=query_body)
    return additional_relevant_terms
```

- Mapping occurs after preprocessing, specifically following the removal of noise, ensuring that untagged terms are devoid of unnecessary elements such as stop words.
- This streamlined approach not only minimizes unnecessary mappings but also leverages the efficiency of a ranked list stored as an Elasticsearch document.

Infrastructure changes

1. **Dependency Management:**
    - Installation of spaCy: To support the implementation of spaCy for efficient text processing, the platform's infrastructure will require the installation and management of the spaCy library.
    - SpaCy Model Download: The infrastructure will need to include the downloading of the appropriate spaCy model, such as "en_core_web_sm," for consistent and reliable text processing.
2. **Data Storage for Keyword Ranking:**
    - During the corpus preprocessing, a keyword ranking list is generated using TF-IDF calculations. To ensure this valuable data is available in production, infrastructure changes will include appropriate storage mechanisms.
    - The biologically significant terms and their associated scores (keyword ranking list) will be stored as an elastic document for seamless access during query processing.

Future Aspects:

1. **Multi-Repository Support:** Extending advanced query parsing to repositories like TCGA, HPA, and cBioPortal.
2. **Diverse Data Types:** Expanding to various data types, including genomics, proteomics, and clinical data